

Introduction to Scikit-learn

a Python Machine Learning Package

Part II: Comparing ML Models



Titli Sarkar

Computational Scientist

Titli.Sarkar@nih.gov

Advanced Biomedical & Computational Science (ABCS)



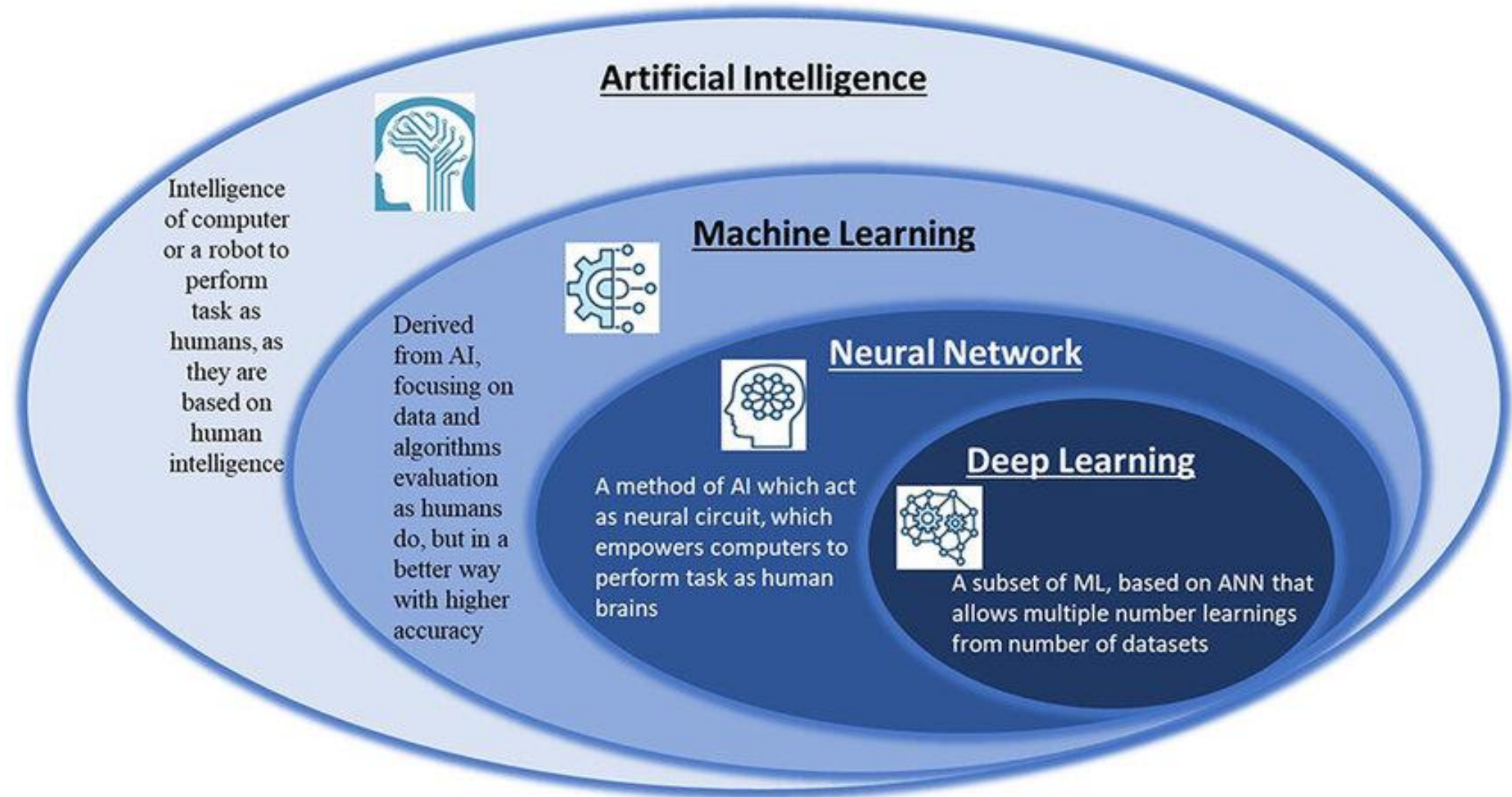
Features	Python	R
Purpose of Existence	☐ Open-source ☐ Beginner-friendly ☐ Multi-Purpose ☐ Scalable	☐ Open source ☐ Best for data visualization ☐ Complex statistical analysis ☐ Data analysis tasks
Usability	☐ Simplicity and code readability ☐ Smooth learning curve	☐ Steep learning curve for developers who do not have prior statistical language programming skills.
Performance	☐ Faster for computationally intensive tasks and large-scale data processing, especially when optimized. ☐ Better memory usage for large datasets.	☐ Relatively slower than python for large datasets and complex operations. ☐ Optimization function like <i>lapply()</i> can improve performance.
Data Manipulation	☐ <i>Pandas</i> -> data cleaning, transformation, and manipulation (Table as Dataframe).	☐ Excels with data frames and offers various packages for data reshaping and manipulation, often favored by <i>statisticians</i> .
Data Visualization	☐ <i>Matplotlib, Seaborn, and Plotly</i> -> extensive visualization options.	☐ <i>ggplot2</i> -> widely-used powerful library for creating high-quality, customizable statistical graphics.
Statistical Analysis	☐ <i>NumPy, SciPy, and StatsModels</i> for numerical and statistical computing, with a strong focus on machine learning.	☐ Designed specifically for statistical computing , with a vast array of built-in statistical functions and specialized packages for various analyses.
Machine Learning	☐ Dominant in machine learning and deep learning with libraries like <i>scikit-learn, TensorFlow, and PyTorch</i> .	☐ Provides libraries like <i>Caret and MLR</i> for machine learning, though its ecosystem is generally more focused on traditional statistical modeling.

What should I choose: R or Python?

Summary:

- **R** is often preferred by statisticians and researchers for its specialized statistical capabilities and exceptional data visualization tools (especially ggplot2).
- **Python** is a more versatile, general-purpose language favored for its ease of learning, extensive machine learning libraries, and strong capabilities in data engineering and deployment.
- The choice between Python and R often depends on the specific project requirements, team expertise, and industry standards. Many data professionals find value in learning both languages to leverage their respective strengths.

Era of AI



Why Scikit-learn?

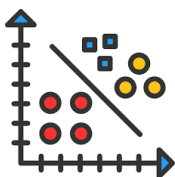
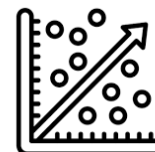
What is Scikit-learn?

- It is an *open-source* Machine Learning library for the Python programming language.
- It provides a wide range of **built-in** metrics for algorithms and tools for various machine learning tasks, including:



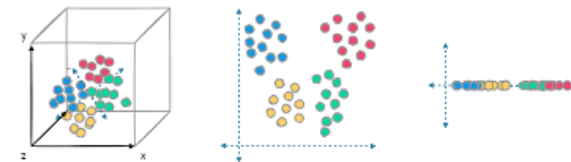
Classification: Identifying which category an object belongs to (e.g., spam detection, image recognition).

Regression: Predicting a continuous-valued attribute associated with an object (e.g., stock prices, drug response).



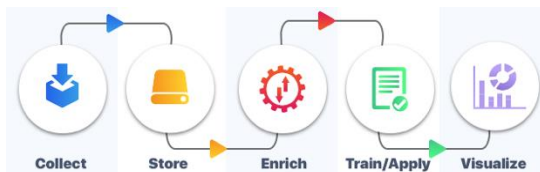
Clustering: Automatic grouping of similar objects into sets (e.g., customer segmentation).

Dimensionality Reduction: Reducing the number of random variables to consider (e.g., for visualization or increased efficiency).



Model Selection: Comparing, validating, and choosing parameters and models (e.g., grid search, cross-validation).

Preprocessing: Feature extraction and normalization (e.g., transforming input data for use with machine learning algorithms).



REMOVING NOISE



STANDARDIZATION



COMBINING SOURCES



SIMPLIFICATION



HANDLING MISSING VALUES

Built on top of other popular Python libraries like *NumPy*, *SciPy*, and *Matplotlib*, scikit-learn offers a user-friendly and consistent API, making it a popular choice for both beginners and experienced practitioners in the field of machine learning and data science.

Scikit-learn vs. other Python ML libraries



scikit-learn vs. Pandas

- **Pandas** is ideal for data manipulation and analysis.
- **Scikit-learn** is perfect for machine learning tasks.
- In practice, these libraries often work together – NumPy for calculations, Pandas for data preparation, and Scikit-learn for model building.

Pandas Basic Operations

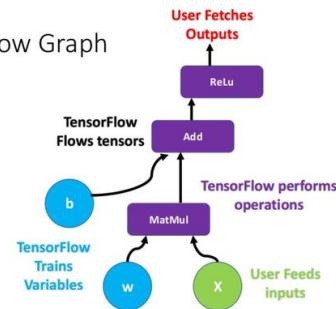
Pandas simplifies data handling by enabling efficient preprocessing, cleaning, transformation, and visualization.



scikit-learn vs. TensorFlow

- **Scikit-Learn** is generally considered *better* for beginners due to its simplicity and ease of use.
- **TensorFlow** has a steeper learning curve and is more suitable for individuals with prior experience or those specifically interested in deep learning.

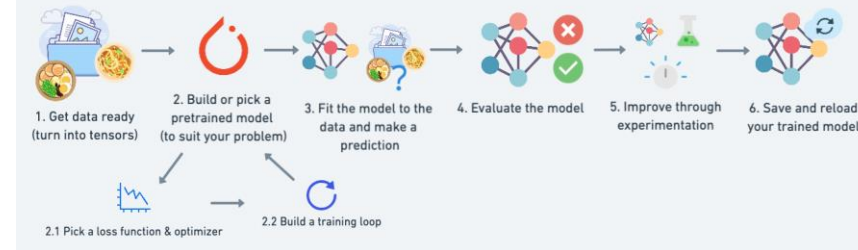
TensorFlow Graph



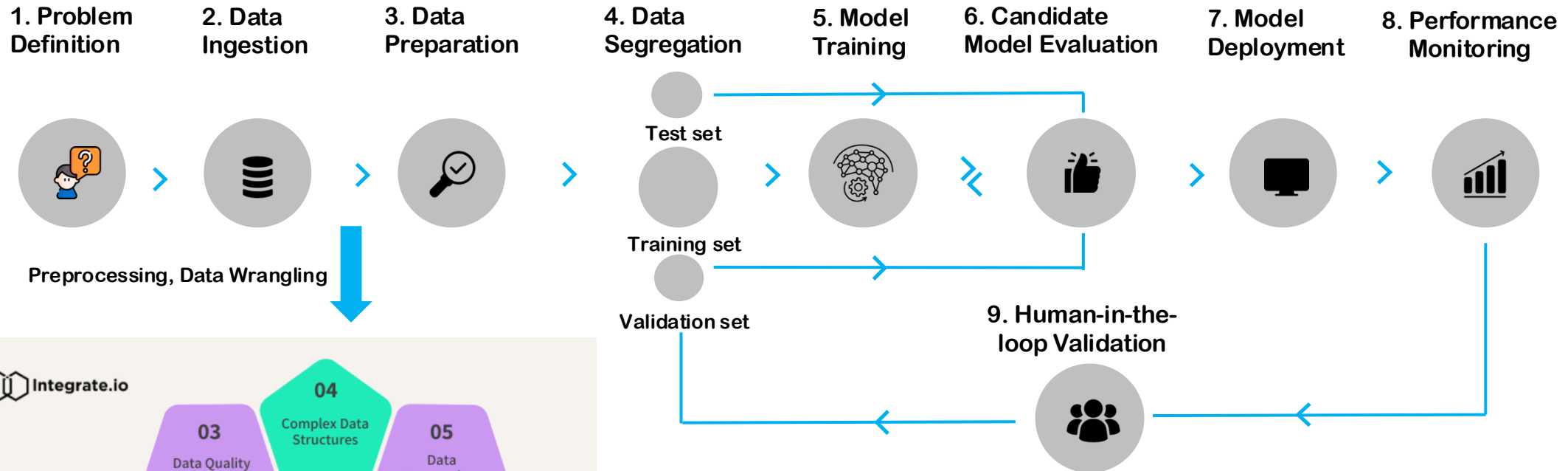
scikit-Learn vs. PyTorch

- **Sklearn** is built on top of Python libraries like NumPy, SciPy, and Matplotlib, and is simple and efficient for data analysis, efficient for machine learning.
- **PyTorch** is designed for deep learning projects, large-scale applications, custom neural network architectures, and when leveraging GPU acceleration is necessary.

A PyTorch Workflow



Basic Machine Learning pipeline

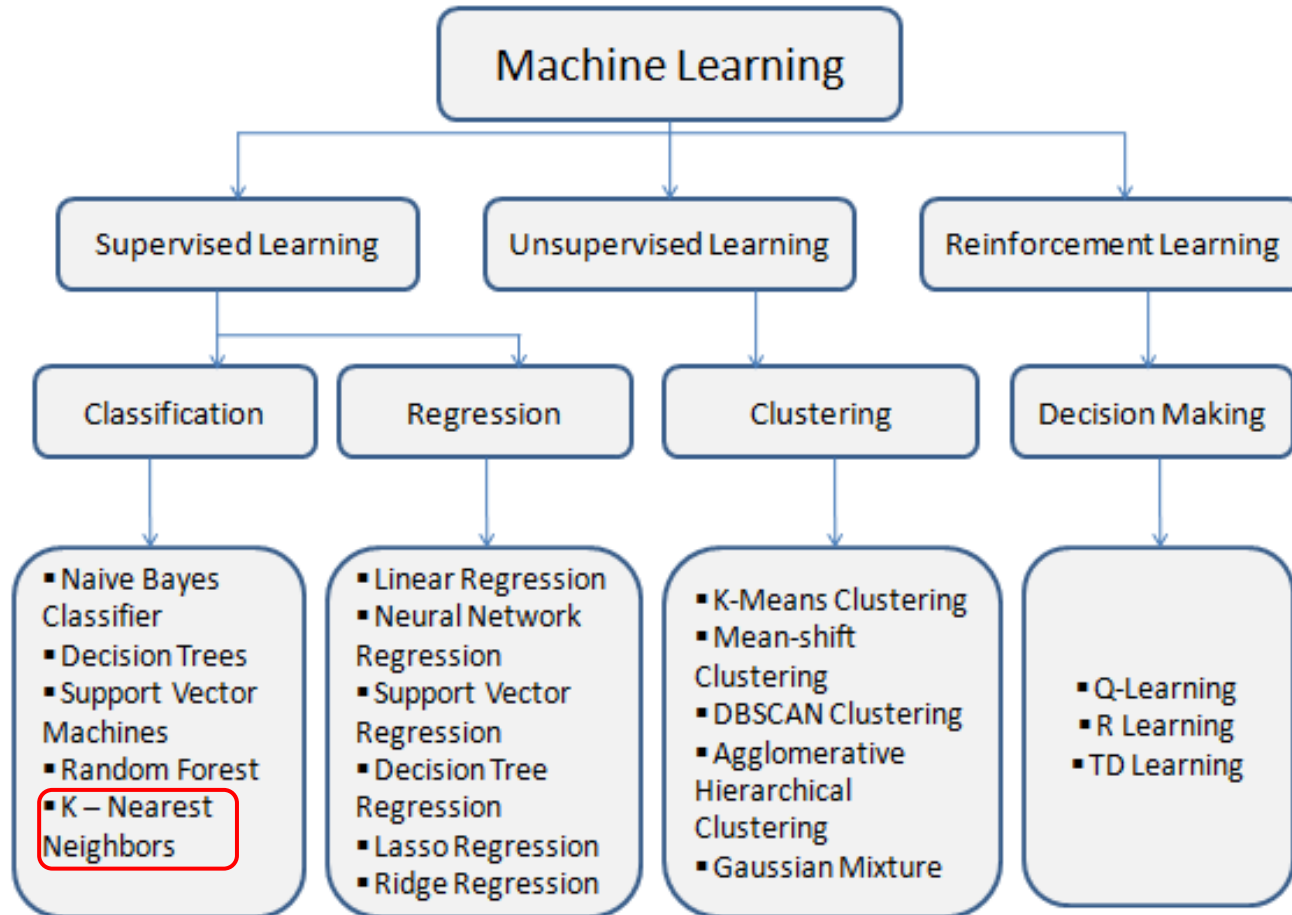


Machine Learning Model Training:

Iterative Process: an algorithm learns from data to make predictions or perform specific tasks.

- First, choose an appropriate algorithm.
- Second, feed the algorithm a dataset (model training),
- Next, through iterative adjustments to its internal parameters (hyperparameter tuning), refine the algorithm's ability to recognize patterns and make accurate predictions (minimize loss).

Machine Learning Model selection



Useful Resources:

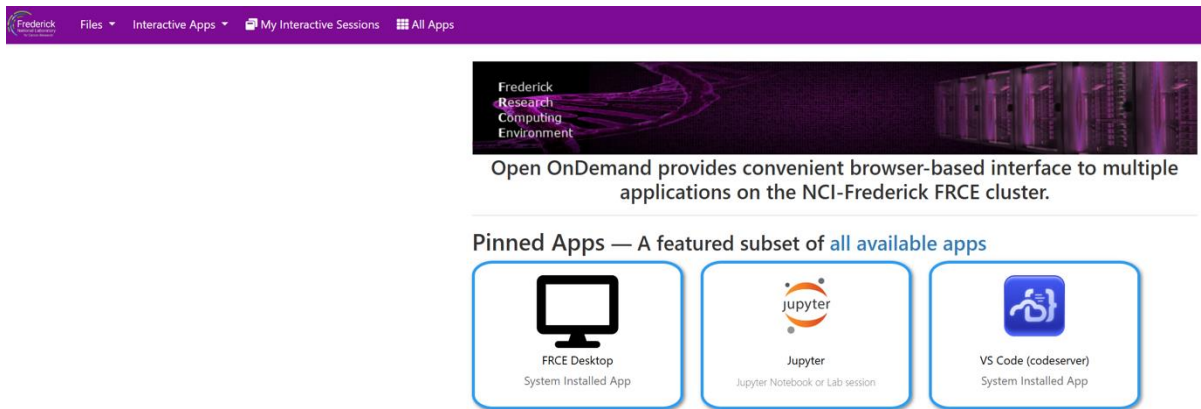
- [Getting Started with HPC OnDemand: A New Interface for NIH HPC Users:](#) Jonathan Goodson (CIT)
- [BTEP Coding Club: Submitting Scripts to the Biowulf Batch System:](#) Joe Wu (BTEP)
- [Documenting Data Analysis with Jupyter Lab:](#) Joe Wu (BTEP)
- <https://bioinformatics.ccr.cancer.gov/btep/resources/scientific-software/>: BTEP softwares
- [Decision Trees, Survival Trees, and Random Forest:](#) Brian Luke (ABCS)
- [Decision Trees, Survival Trees, and Random Forest: Practical Examples with R Programming:](#) Brian Luke (ABCS)
- [Correlation and Regression Methods in Statistical Analysis:](#) Alexander Mitrophanov (DMS)
- [Introduction to Clustering :](#) Brian Luke (ABCS)
- [Clustering with R and Rstudio :](#) Brian Luke (ABCS)
- [Survival Analysis in R Using the survival Package:](#) Duncan Donohue (DMS)
- [Introduction to Data Exploration:](#) Duncan Donohue (DMS)
- [Missing Values and Data Transformations:](#) Duncan Donohue (DMS)

How can I get a hands-on?

Option 1 : Download and install Anaconda3 to your computer (large space), which comes with Python, Spider (run .py scripts) and Jupyter (run .ipynb notebooks). **Not Recommended** or, save yourself from the headache of installation!

Option 2 : NIH HPC have all of it installed and readily available for users at both Biowulf and FRCE via Open Ondemand.

- <https://ondemand.ncifcrf.gov>

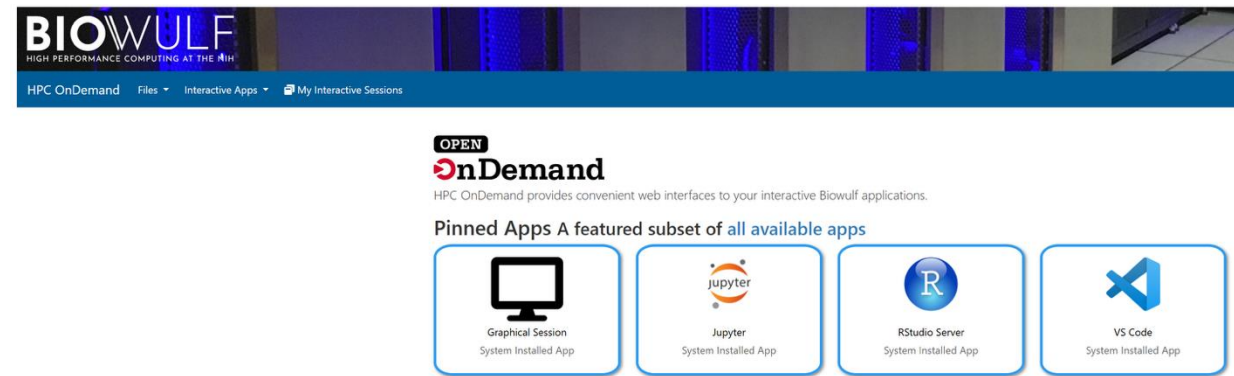


FRCE

***Important!**

1. You need to navigate to the directory where your data and code are located. Jupyter will start from the directory you selected at the startup.
2. Please be respectful to others and be mindful while requesting for resources, Do NOT ask for resources you don't need.
 - Jupyter notebook is an interactive way of Python scripting, mostly used for learning purpose. Almost never you need a GPU node for running a Jupyter notebook, so, please do not ask.
 - Also, be careful about amount of memory and CPU cores.
3. Do NOT forget to relinquish resources after you are done.
 - You need to come back to you session homepage and 'DELETE' the session once you are finished.

- <https://hpcondemand.nih.gov/>



Biowulf

Dataset

- <https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset> or,
- https://www.kaggle.com/code/mahmoudbahnasy29/diabetes?select=diabetes_prediction_dataset.csv
- **diabetes_prediction_dataset.csv** file: contains medical and demographic data of patients along with their diabetes status, whether positive or negative. It consists of various features such as age, gender, body mass index (BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level. The Dataset can be utilized to construct machine learning models that can predict the likelihood of diabetes in patients based on their medical history and demographic details.

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0
...	...	...	...	...	...	...	...	...	...
99995	Female	80.0	0	0	No Info	27.32	6.2	90	0
99996	Female	2.0	0	0	No Info	17.37	6.5	100	0
99997	Male	66.0	0	0	former	27.83	5.7	155	0
99998	Female	24.0	0	0	never	35.42	4.0	100	0
99999	Female	57.0	0	0	current	22.43	6.6	90	0

100000 rows × 9 columns

Information about the Data

```
df.info(verbose=True)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   gender                100000 non-null object
1   age                  100000 non-null float64
2   hypertension          100000 non-null int64
3   heart_disease         100000 non-null int64
4   smoking_history       100000 non-null object
5   bmi                  100000 non-null float64
6   HbA1c_level           100000 non-null float64
7   blood_glucose_level   100000 non-null int64
8   diabetes              100000 non-null int64
dtypes: float64(3), int64(4), object(2)
memory usage: 6.9+ MB
```

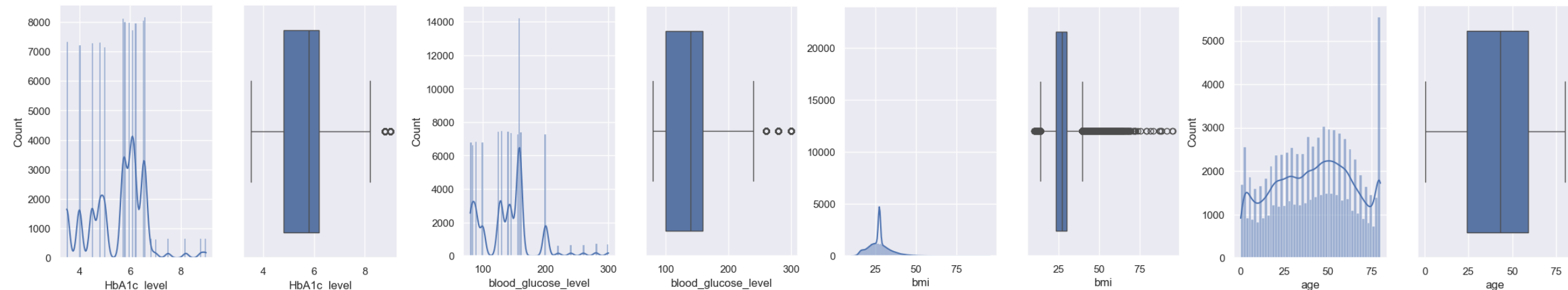
Data Statistics

```
df.describe(include="all").T
```

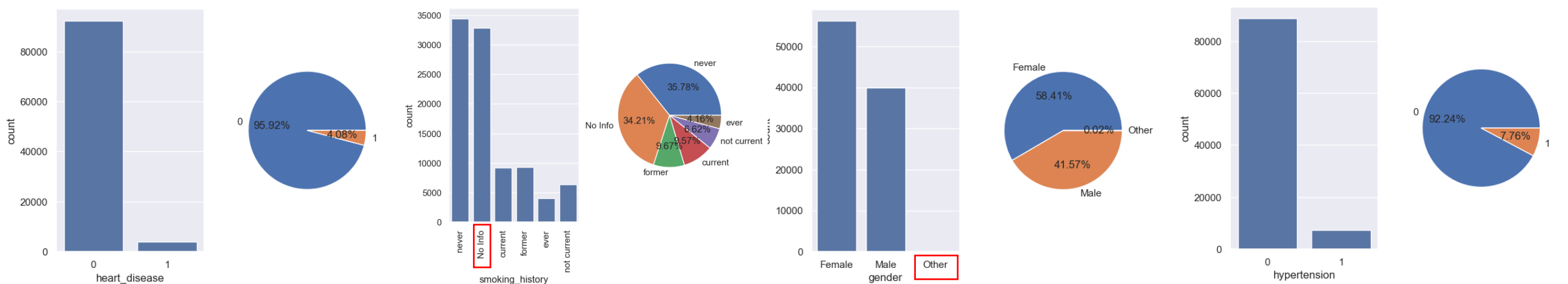
	count	mean	std	min	25%	50%	75%	max
gender	100000.0	1.585340	0.493031	0.00	1.00	2.00	2.00	2.00
age	100000.0	41.885856	22.516840	0.08	24.00	43.00	60.00	80.00
hypertension	100000.0	0.074850	0.263150	0.00	0.00	0.00	0.00	1.00
heart_disease	100000.0	0.039420	0.194593	0.00	0.00	0.00	0.00	1.00
smoking_history	100000.0	1.433770	1.627925	0.00	0.00	1.00	3.00	5.00
bmi	100000.0	27.320767	6.636783	10.01	23.63	27.32	29.58	95.69
HbA1c_level	100000.0	5.527507	1.070672	3.50	4.80	5.80	6.20	9.00
blood_glucose_level	100000.0	138.058060	40.708136	80.00	100.00	140.00	159.00	300.00
diabetes	100000.0	0.085000	0.278883	0.00	0.00	0.00	0.00	1.00

Univariate Analysis

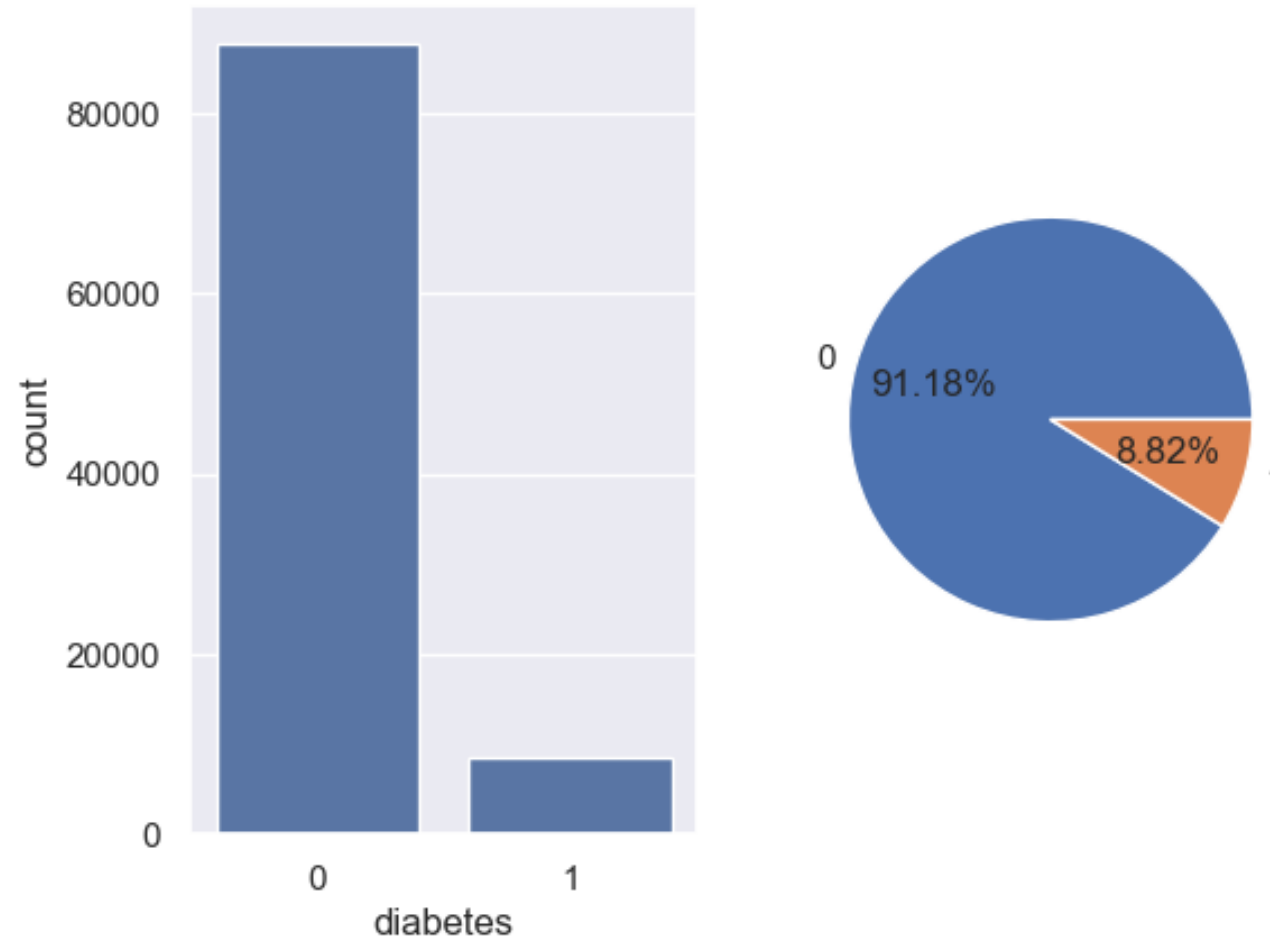
First, let's take a look at the data visually ...



Pie plot for categorical variables



Target Variable 'diabetes' shows Data Imbalance



Bivariate Analysis

- Check co-relation between features
- hue =diabetes shows percentage of diabetes positive or negative patients within each feature variable

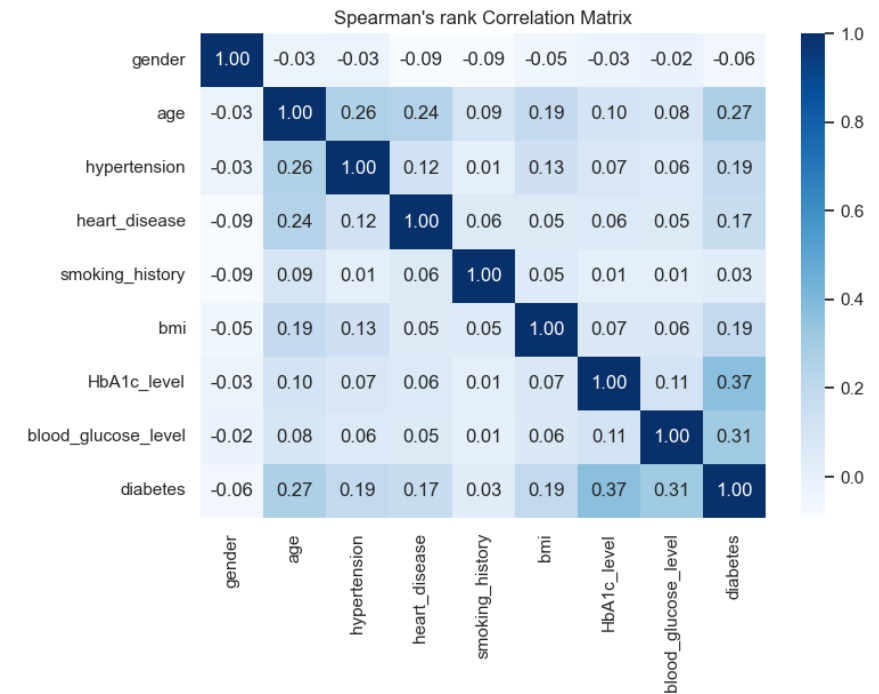
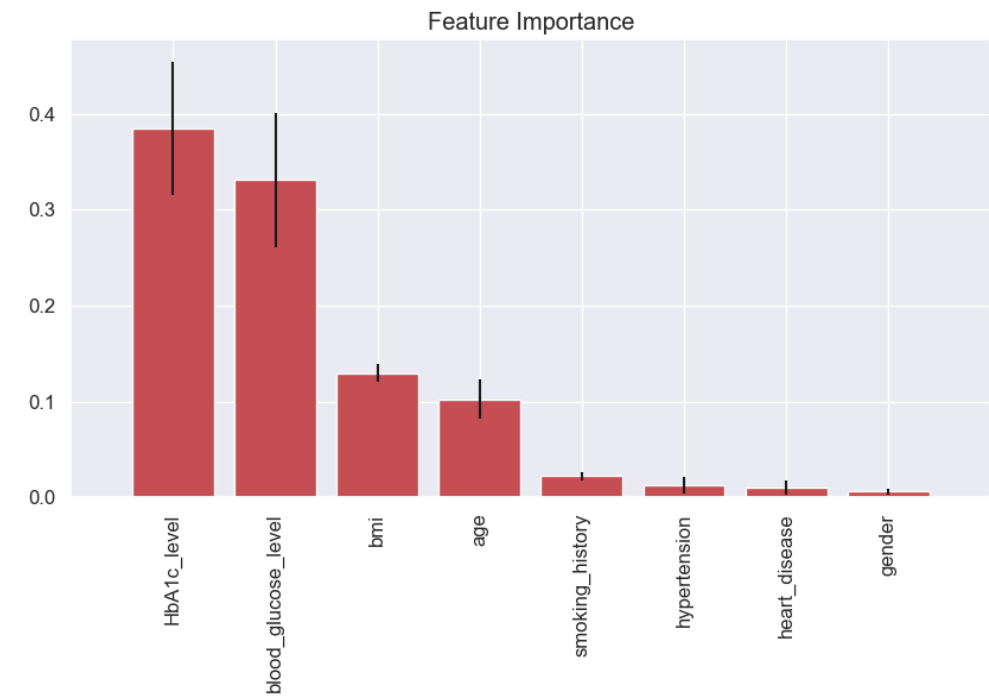


Preprocessing

- 3854 duplicates -> drop them, keeping the first.
- No null values. Consider zeros for some columns as null and remove them.
- Standardization (min-max Normalization or Scaling) to rescales features between 0 and 1.
- Distance-based Machine Learning algorithms can only read numerical values, therefore, convert categorical columns to numeric.

```
df_min_max_scaled.describe(include="all").T
```

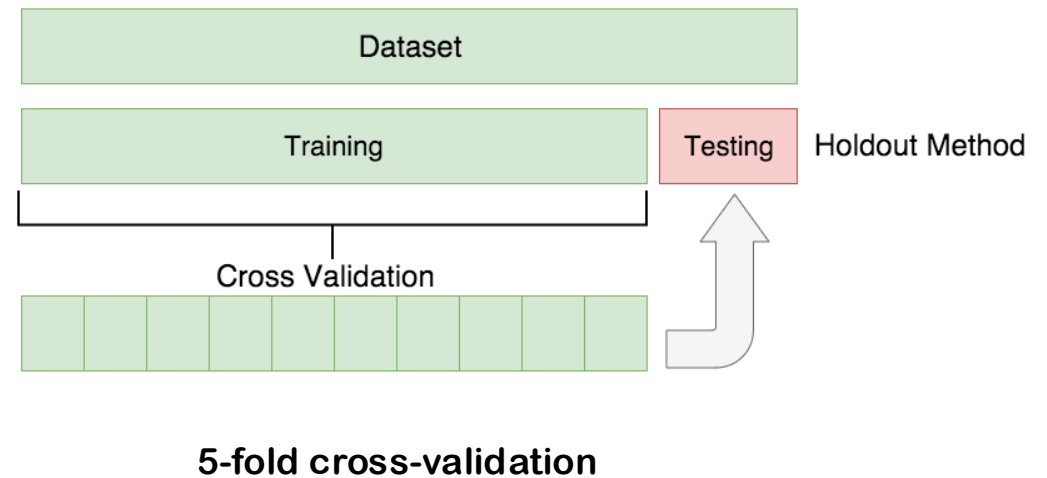
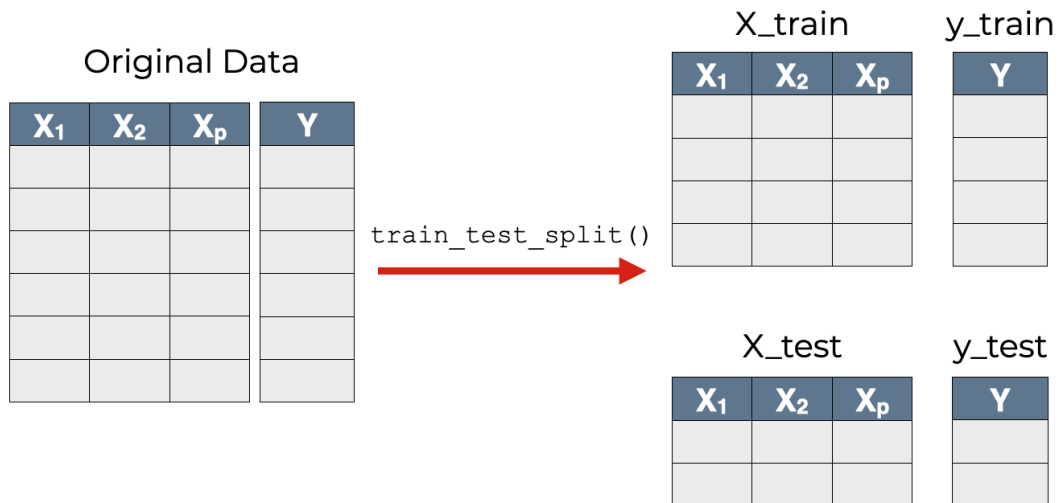
	count	mean	std	min	25%	50%	75%	max
gender	100000.0	0.792670	0.246515	0.0	0.500000	1.000000	1.000000	1.0
age	100000.0	0.523096	0.281742	0.0	0.299299	0.537037	0.749750	1.0
hypertension	100000.0	0.074850	0.263150	0.0	0.000000	0.000000	0.000000	1.0
heart_disease	100000.0	0.039420	0.194593	0.0	0.000000	0.000000	0.000000	1.0
smoking_history	100000.0	0.286754	0.325585	0.0	0.000000	0.200000	0.600000	1.0
bmi	100000.0	0.202040	0.077460	0.0	0.158964	0.202031	0.228408	1.0
HbA1c_level	100000.0	0.368638	0.194668	0.0	0.236364	0.418182	0.490909	1.0
blood_glucose_level	100000.0	0.263900	0.185037	0.0	0.090909	0.272727	0.359091	1.0
diabetes	100000.0	0.085000	0.278883	0.0	0.000000	0.000000	0.000000	1.0



Data splitting

```
# split data using scikit-Learn
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 42)
```



Fundamental step to ensure that the model's performance is accurately evaluated and to prevent **overfitting** (model performs well on the training data, but poorly performs on unseen test data).

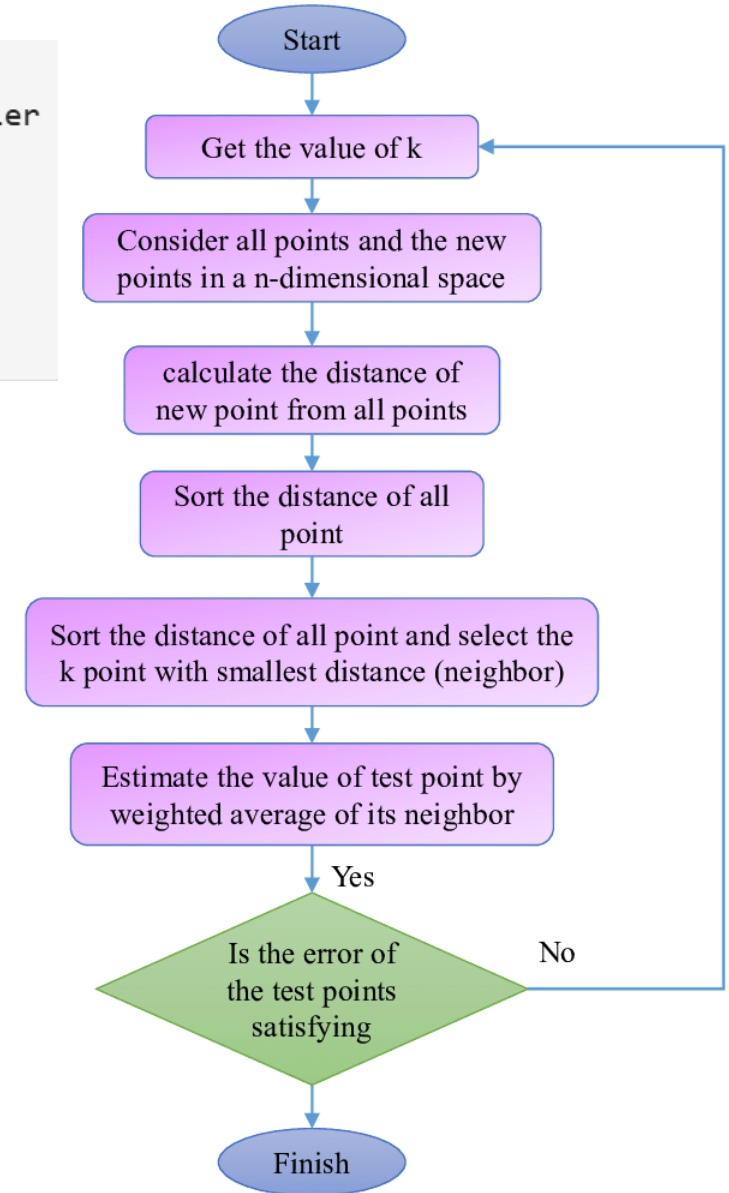
Classification model : k Nearest Neighbors (knn)

```
# use in-built knn ML model from scikit-learn
from sklearn.neighbors import KNeighborsClassifier

k = best_k
model_knn = KNeighborsClassifier(n_neighbors=k)
model_knn.fit(X_train, y_train)
pred_knn = model_knn.predict(X_test)
```

Key Aspects:

- **Supervised** ML algorithm,
- Used for both Classification and Regression
- **Non-parametric**: It makes no assumptions about the underlying data distribution.
- **Distance-based**: Its performance heavily relies on the chosen distance metric.
- *** Sensitive to 'k'**: The choice of 'k' is crucial; a small 'k' can lead to overfitting, while a large 'k' can lead to underfitting. Cross-validation is often used to find an optimal 'k'.
- **Computational cost**: For large datasets, calculating distances to all training points can be computationally expensive during prediction.
- Considered as a "lazy" learning algorithm because it does not build a model during the training phase but instead stores the entire training dataset. All computation is deferred until a new data point needs to be classified or its value predicted.



How to choose the value of **k**?

- Crucial step for achieving good model performance. While there's NO single "best" method, several approaches are commonly employed.



Cross-Validation

Most robust method.

Iterate through a range of **k** values (e.g., from 1 to a reasonable upper limit):

- For each **k**, perform **k-fold cross-validation** on your training data.
- Calculate the average performance metric (e.g., **accuracy, F1-score**) for each **k**.
- Select the **k** that yields the best average performance.

* Important Considerations:

- **Odd **k** for Classification:** When dealing with binary or multi-class classification, choosing an **odd value** for **k** is generally preferred to prevent ties when determining the majority class among the neighbors.
- **Computational Cost:** Larger **k** values increase the computational cost of finding neighbors.
- **Bias-Variance Trade-off:** A small **k** can lead to high variance (overfitting to noise), while a large **k** can lead to high bias (*oversmoothing and underfitting*). The goal is to find a **k** that balances this trade-off.



Domain Knowledge

Consider the nature of your data and problem.

If you expect local patterns to be very distinct, a smaller **k** might be appropriate.

If the data is noisy, a larger **k** can help to smooth out the decision boundaries.

Model Evaluation

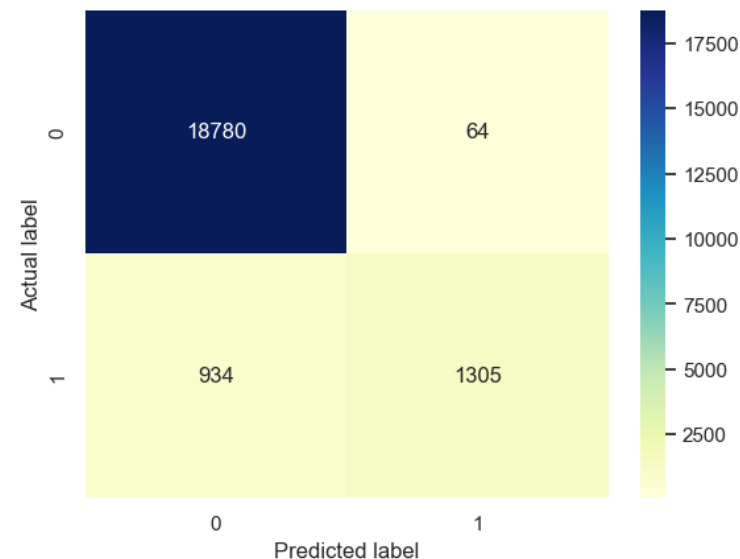
```
# Evaluate the model using scikit-Learn metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import mean_squared_error, r2_score, roc_auc_score
from sklearn.metrics import confusion_matrix, classification_report
```

```
pred_knn = model_knn.predict(X_test)
```

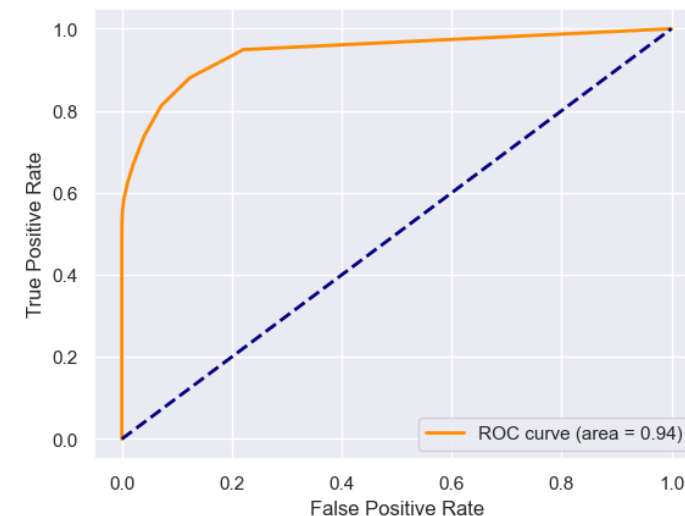
```
accuracy = accuracy_score(y_test, pred_knn)
precision = precision_score(y_test, pred_knn)
recall = recall_score(y_test, pred_knn)
auc = roc_auc_score(y_test, pred_knn)
cr = classification_report(y_test, pred_knn)
cm = confusion_matrix(y_test, pred_knn)
```

	precision	recall	f1-score	support
0.0	0.95	1.00	0.97	18844
1.0	0.95	0.58	0.72	2239
accuracy			0.95	21083
macro avg	0.95	0.79	0.85	21083
weighted avg	0.95	0.95	0.95	21083

Confusion matrix



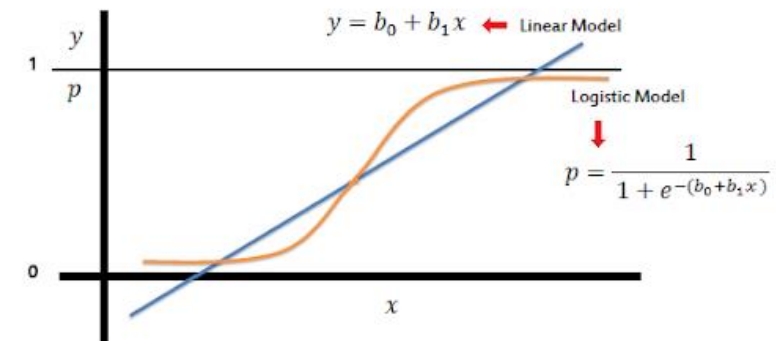
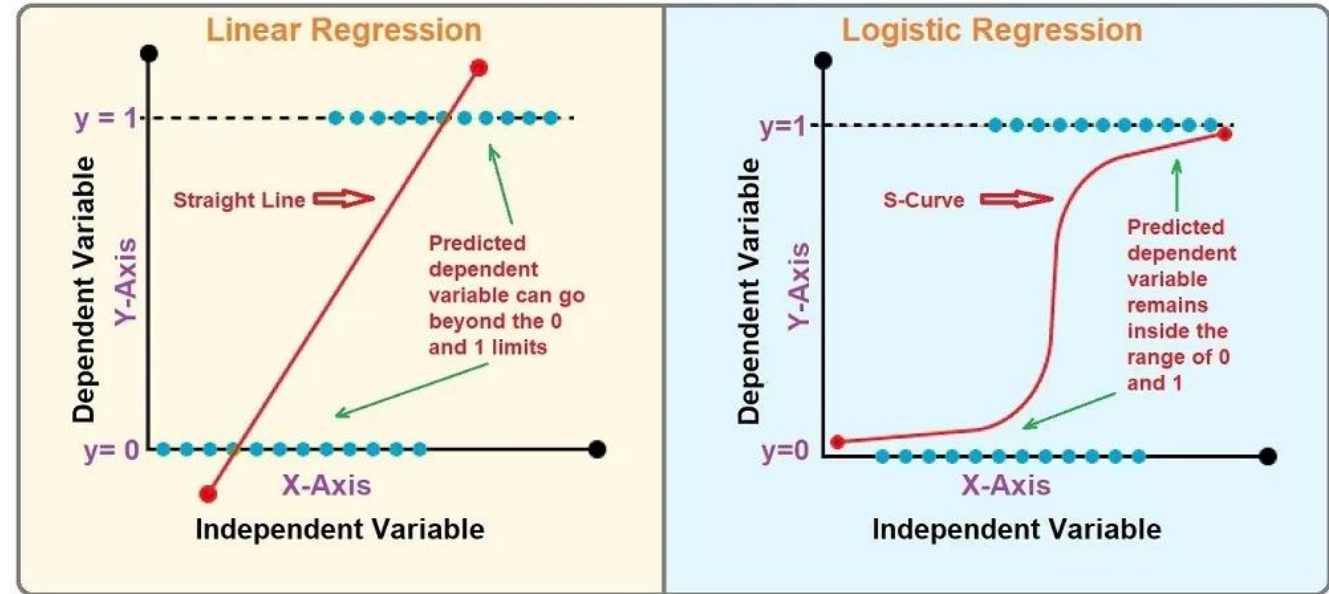
ROC curve



Classification model : Logistic Regression

Key Aspects:

- **Binary Outcome:** The dependent variable has only two possible categories (e.g., Pass/Fail, Disease/No Disease).
- **Probability:** It predicts the probability (P) that an instance belongs to the "positive" class (e.g., $P(\text{Diabetis})$).
- **S-Shaped Curve (Sigmoid):** A logistic function transforms the linear combination of inputs into a probability, ensuring the output stays between 0 and 1.
- **Classification:** A threshold (e.g., 0.5) is applied to the probability; if $P > 0.5$, it's classified as one class, otherwise the other.
- **Predictors:** Can handle multiple independent variables (features) like age, weight, or test scores to make predictions.



Model Evaluation

```
# Import module from scikit-learn
from sklearn.linear_model import LogisticRegression

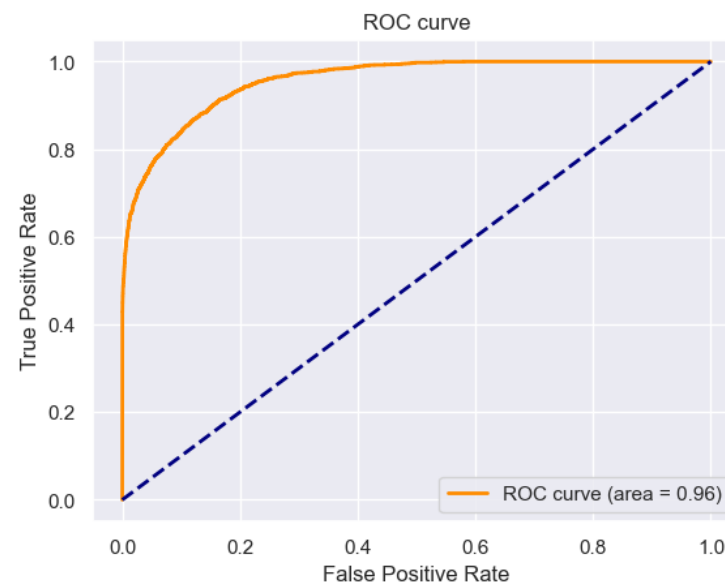
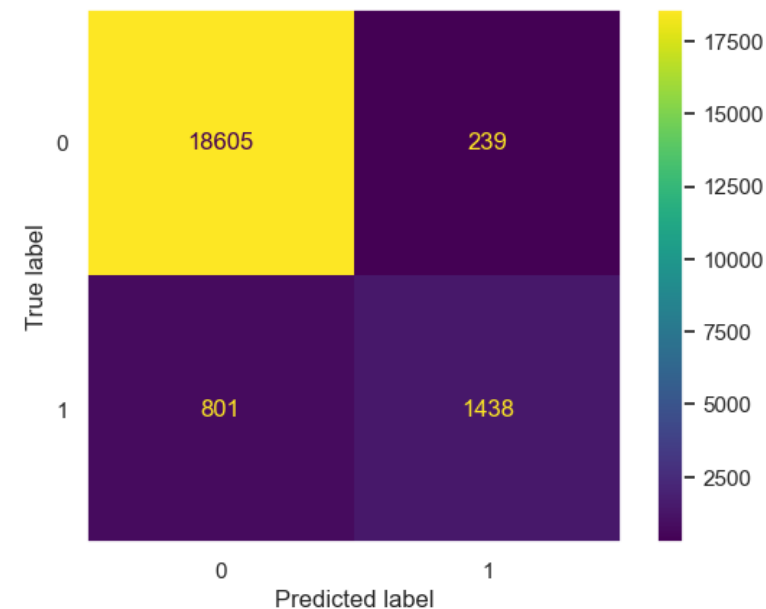
## Initialize and train the Logistic Regression model
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)

# Make predictions
pred_lr = model_lr.predict(X_test)

# Model Evaluation
print(accuracy_score(y_test, pred_lr))
print(classification_report(y_test, pred_lr))
```

Classification Report:

	precision	recall	f1-score	support
0.0	0.96	0.99	0.97	18844
1.0	0.86	0.64	0.73	2239
accuracy			0.95	21083
macro avg	0.91	0.81	0.85	21083
weighted avg	0.95	0.95	0.95	21083



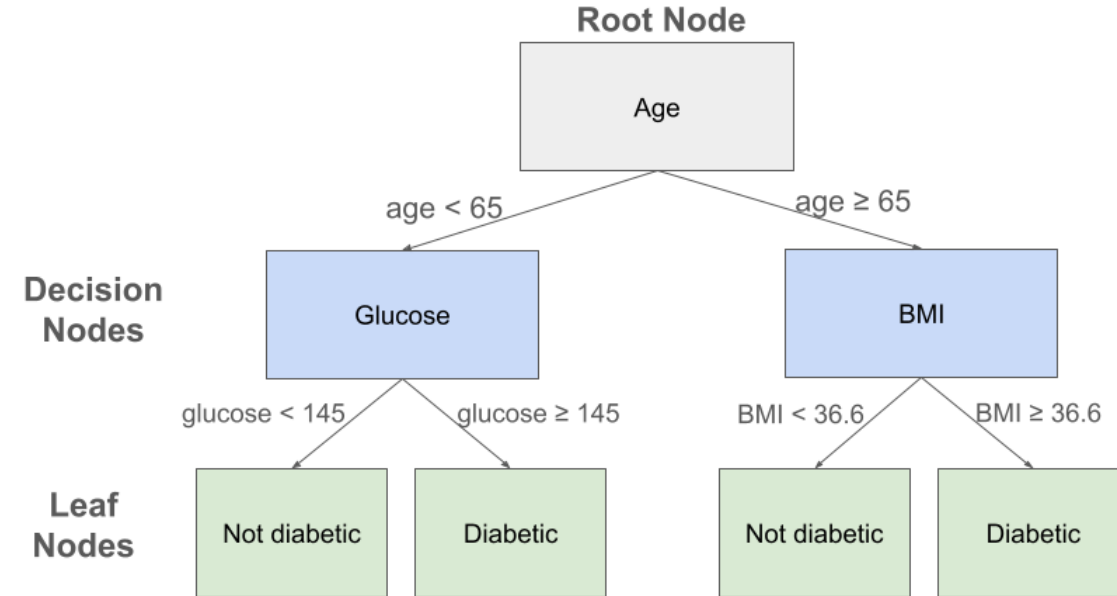
Classification model : Decision Tree

A decision tree in Python is a supervised machine learning algorithm used for both classification and regression tasks.

It operates by building a *hierarchical, flowchart-like* model in the form of a tree structure, to sort data into predefined classes.

Key Aspects:

- **Root Node:** The topmost node in the tree, representing the initial decision or feature test.
- **Internal Nodes:** Nodes that represent a test on a feature and have branches leading to other nodes.
- **Leaf Nodes:** Terminal nodes that represent the final outcome or prediction.
- **Recursive Splitting:** The algorithm employs a greedy, top-down approach to split the data. At each node, it selects the best feature and a corresponding threshold to partition the data.
- **Splitting Criteria:** The "best" split is determined by metrics that measure the purity or homogeneity of the resulting subsets, such as:
 - **Gini Impurity:** Measures the probability of misclassification if a class label is randomly chosen from the node's distribution. A lower Gini value indicates higher purity.
 - **Entropy/Information Gain:** Entropy measures the amount of uncertainty or randomness in the data. The goal is to maximize the information gain, which is the reduction in entropy after a split.



* **Stopping Conditions:** maximum depth, #of minimum samples in a node or leaf, or when the impurity decrease is below a certain threshold.

Instability: Small variations in the input data can lead to a completely different tree structure.

Bias: Can be biased towards dominant classes if the dataset is imbalanced.

Model Evaluation

```
# import module from scikit-learn
from sklearn.tree import DecisionTreeClassifier

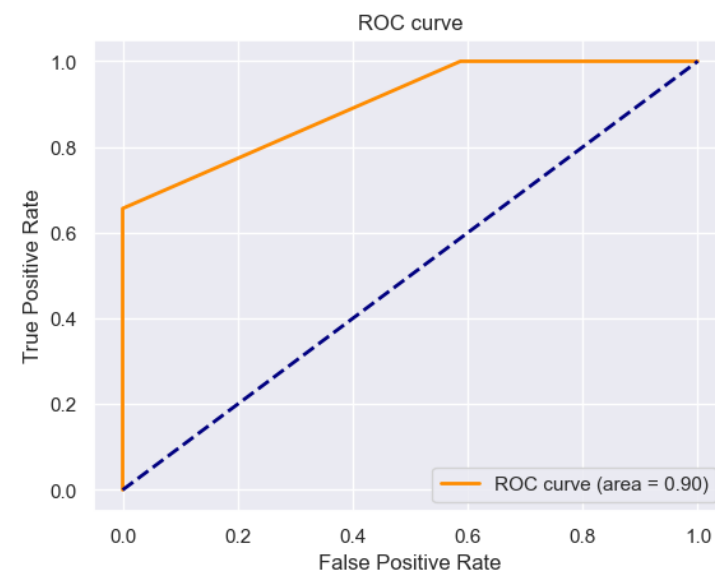
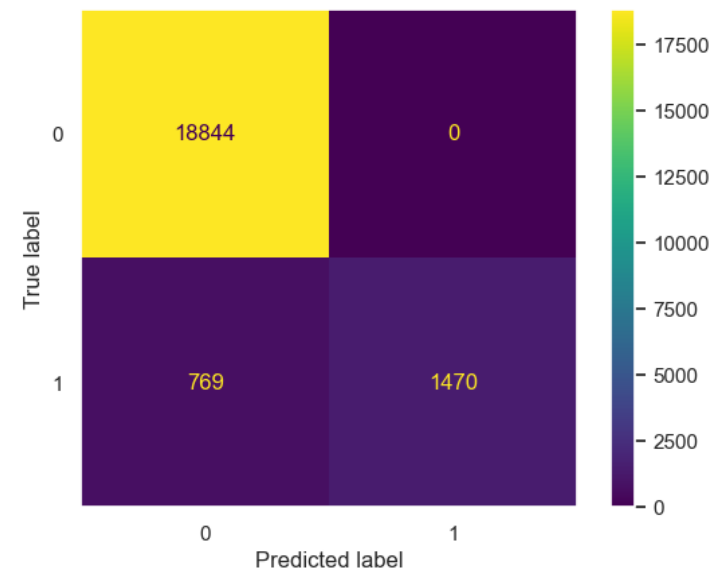
# Instantiate the Decision Tree Classifier
model_dt = DecisionTreeClassifier(max_depth=3, random_state=42)

# Train the model
model_dt.fit(X_train, y_train)

# Make predictions
pred_dt = model_dt.predict(X_test)

# Evaluate the model
print(accuracy_score(y_test, pred_dt))
print(classification_report(y_test, pred_dt))
```

Classification Report:				
	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	18844
1.0	1.00	0.66	0.79	2239
accuracy			0.96	21083
macro avg	0.98	0.83	0.89	21083
weighted avg	0.96	0.96	0.96	21083

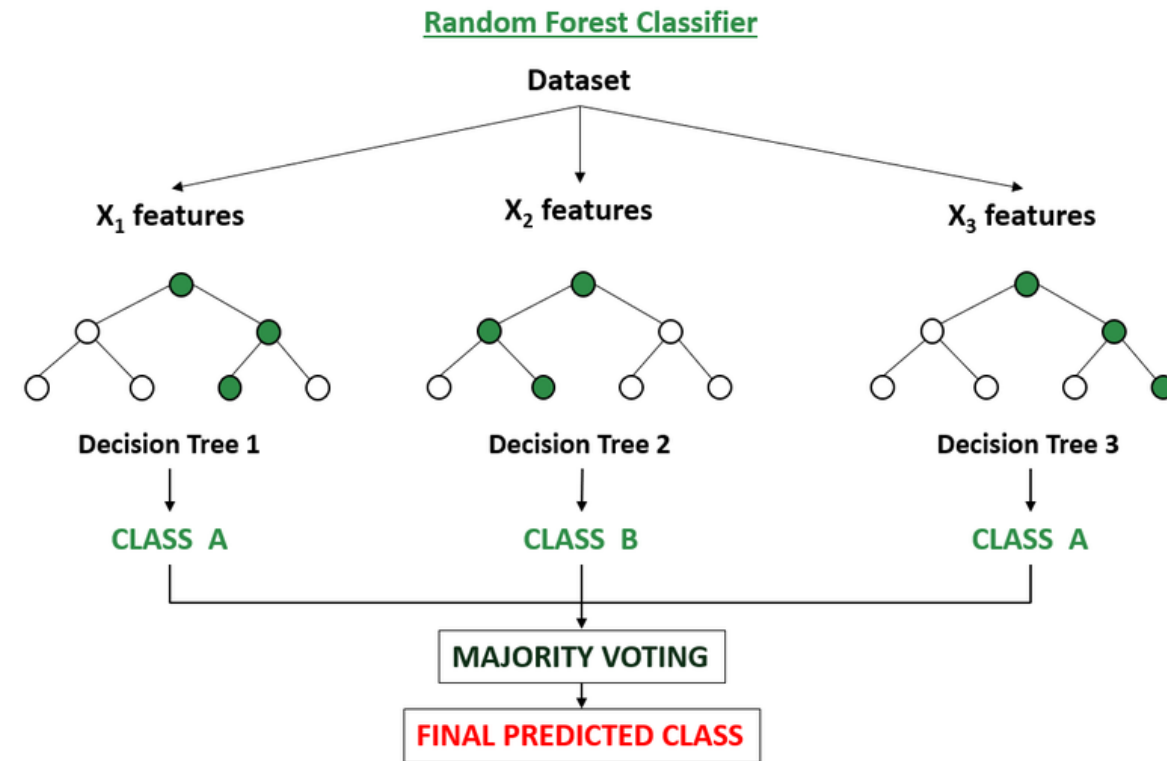


Classification model : Random Forest

An ensemble machine learning method that builds and combines multiple individual decision trees to produce a more accurate and stable final prediction. It is widely used for its robustness against overfitting, ability to handle large datasets, and built-in feature importance estimation, but it is computationally expensive.

Key Aspects:

- **Bootstrapping:** The algorithm creates multiple subsets of the original training data by randomly sampling with replacement. Each decision tree in the forest is trained on one of these unique, random data samples.
- **Feature Randomness:** When growing each tree, instead of considering all available features for the best possible split at each node, the algorithm only considers a random subset of features. This ensures that the individual trees are not highly correlated with each other.
- **Building the Trees:** Each decision tree is grown to its largest extent possible without pruning, ensuring a low bias for individual trees.
- **Majority Voting (Bagging):** For a classification task, each tree in the forest "votes" for a class label. The final output of the random forest model is the class that receives the majority of the votes.



Model Evaluation

```
# Import module from scikit-learn
from sklearn.ensemble import RandomForestClassifier

# Instantiate the Random Forest Classifier
model_rf = RandomForestClassifier(n_estimators=100, random_state=42) # n_estimators is number of trees

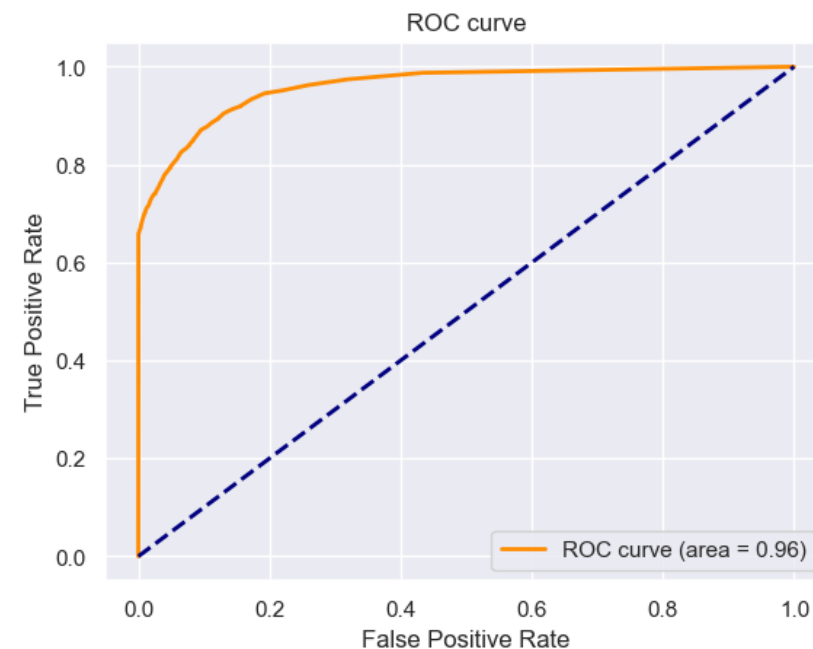
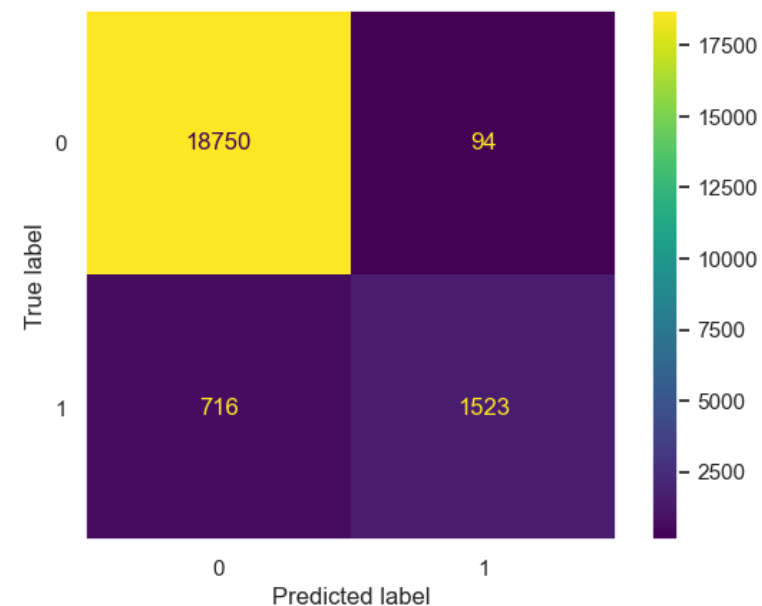
# Train the model
model_rf.fit(X_train, y_train)

# Prediction
pred_rf = model_rf.predict(X_test)

# Evaluate the model
print(accuracy_score(y_test, pred_rf))
print(classification_report(y_test, pred_rf))
```

Classification Report:

	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	18844
1.0	0.94	0.68	0.79	2239
accuracy			0.96	21083
macro avg	0.95	0.84	0.88	21083
weighted avg	0.96	0.96	0.96	21083



Best model selection

K-nearest neighbor:

	precision	recall	f1-score	support
0.0	0.95	1.00	0.97	18844
1.0	0.95	0.58	0.72	2239
accuracy			0.95	21083
macro avg	0.95	0.79	0.85	21083
weighted avg	0.95	0.95	0.95	21083

Logistic regression:

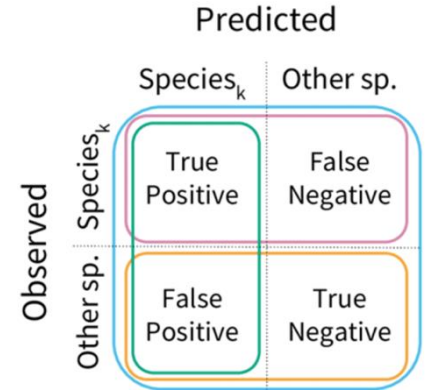
	precision	recall	f1-score	support
0.0	0.96	0.99	0.97	18844
1.0	0.86	0.64	0.73	2239
accuracy			0.95	21083
macro avg	0.91	0.81	0.85	21083
weighted avg	0.95	0.95	0.95	21083

Decision Tree

	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	18844
1.0	1.00	0.66	0.79	2239
accuracy			0.96	21083
macro avg	0.98	0.83	0.89	21083
weighted avg	0.96	0.96	0.96	21083

Random Forest:

	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	18844
1.0	0.94	0.68	0.79	2239
accuracy			0.96	21083
macro avg	0.95	0.84	0.88	21083
weighted avg	0.96	0.96	0.96	21083



■ Accuracy = $\frac{TP + TN}{TP + TN + FP + FN}$
■ Specificity = $\frac{TN}{TN + FP}$
■ Precision = $\frac{TP}{TP + FP}$
■ Recall = $\frac{TP}{TP + FN}$

- All the models make prediction with high accuracy. But, accuracy is not the best and only one parameter for model selection.
- You **do not want to send home any patient with diabetes as she was healthy**. From this perspective, you should choose the model with recall close to 1. Since, Random Forest has highest recall = 0.68 for Diabetic class, you may want to decide for the Random Forest classifier. You *want to choose the largest recall from as first priority* all the considered classifiers. Also, if you want to combine the ROC curve, Random Forest have highest chance of success. But, keep in mind that it is expensive, for larger datasets, you may want to trade-off.
- Adminstrating this model, **you would send home the lowest possible number of patients with diabetes at the cost of reexamining greater number of healthy individuals**.
- Selecting best model depends on your data and purpose. For smaller datasets, Decision tree may generate faster and accurate decision, for larger dataset, Random Forest often works better.

Conclusion

- Keep in mind how important is to explore your data before modeling. Specifically, there are more possibilities how missing values can be recorded. To see a greater picture of your data, it is important to ask yourself common sense questions, e.g. "can someone have 0 blood pressure?".
- Once you identify missing values in your data, you should decide how to deal with them. Would you delete every record with a missing value, would you substitute it with a mean, or would you try to be efficient and keep as much data as possible?
- It is recommended to find feature's importance to make most accurate and realistic predictions.
- Evaluation of the models based on `recall` along with `accuracy` is strongly recommended, because `recall` close to 1 minimizes the number of cases when a patient with diabetes would be classified incorrectly.
- Beyond scikit-learn, deep learning and neural network models are widely used. For example, a visual algorithm can detect cancer when it is trained on pictures of human cells.

Acknowledgement

ABCS

Uma Mudunuri, PhD

Brian Luke, PhD

Joseph Ivanic, PhD

Wojciech Kasprzak, MS

David Bell, PhD

Ernesto Suarez Alvarez, PhD

BTEP

Amy Stonelake, PhD

Alexandra Emmons, PhD

Questions?